**DAY 14 TASKS**

**IO FUNDAMENTALS**

* **JAVA I/O FUNDAMENTALS:**

**Java I/O** (Input and Output) is used to process the input*and*produce the output*.* Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.We can perform **file handling in java** by Java I/O API.

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In java, three streams are created for us automatically. All these streams are attached with console.

* **System.out:**standard output stream
* **System.in:**standard input stream
* **System.err:**standard error stream
* **Read and Write data from the console:**

In Java, there are three different ways for reading input from the user in the command line environment (console).

1. **Using Buffered Reader Class**

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

**Advantages:**

* The input is buffered for efficient reading.

1. **Using Scanner Class**

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it is also can be used to read input from the user in the command line.

**Advantages:**

* Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
* Regular expressions can be used to find tokens.

**Drawback:**

* The reading methods are not synchronized

1. **Using Console Class**

It has been becoming a preferred way for reading user’s input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).  
**Advantages:**

* Reading password without echoing the entered characters.
* Reading methods are synchronized.
* Format string syntax can be used.

**Drawback:**

* Does not work in non-interactive environment (such as in an IDE).

**FileOutputStream:**

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file –

OutputStream f = new FileOutputStream("C:/java/hello")

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows –

File f=new File("C:/java/hello");

OutputStream f = new FileOutputStream(f);

Once you have OutputStream object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

* **Study I/O Package:**

**Java.io Package in Java:**

* This package provides for system input and output through data streams, serialization and the file system. Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package will cause a NullPointerException to be thrown.
* **Java I/O** (Input and Output) is used to process the input and produce the output.
* Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
* We can perform **file handling in java** by Java I/O API.
* **Java file io (NIO):**

The NIO classes are contained in a package called **java.nio** package. It is important to understand that the NIO subsystem does not replace the stream based I/O classes available in **java.io** package, and good working knowledge of stream-based I/O in **java.io** is helpful for understanding NIO.

NIO data transfer is based on buffers. These classes represent a contiguous extent of memory, together with a small number of data transfer operations. Although theoretically these are general-purpose data structures, the implementation may select memory for alignment or paging characteristics, which are not otherwise accessible in Java. Typically, this would be used to allow the buffer contents to occupy the same physical memory used by the underlying operating system for its native I/O operations, thus allowing the most direct transfer mechanism, and eliminating the need for any additional copying. In most operating systems, provided the particular area of memory has the right properties, transfer can take place without using the CPU at all. The NIO buffer is intentionally limited in features in order to support these goals.

There are buffer classes for all of Java's primitive types except Boolean, which can share memory with byte buffers and allow arbitrary interpretation of the underlying bytes.

* **Use path interface to operate on file and directory paths:**

**Path Operations:**

The [Path](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Path.html) class includes various methods that can be used to obtain information about the path, access elements of the path, convert the path to other forms, or extract portions of a path. There are also methods for matching the path string and methods for removing redundancies in a path. This lesson addresses these Path methods, sometimes called syntactic operations, because they operate on the path itself and don't access the file system.

**Creating a Path**:

A Path instance contains the information used to specify the location of a file or directory. At the time it is defined, a Path is provided with a series of one or more names. A root element or a file name might be included, but neither are required. A Path might consist of just a single directory or file name.

You can easily create a Path object by using one of the following get methods from the [Paths](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Paths.html) (note the plural) helper class:

Path p1 = Paths.get("/tmp/foo");

Path p2 = Paths.get(args[0]);

Path p3 = Paths.get(URI.create("file:///Users/joe/FileTest.java"));